OCR A Level Computer Science

H446 – Paper 2

2

Searching algorithms

Unit 12 Algorithms

OCR Oxford Cambridge and RSA
An OCR endorsed teaching and learning tool
PG ONLINE

Objectives

- Write and trace algorithms for linear search and binary search
- Analyse the time complexity of the linear search and binary search algorithms
- Describe and trace the binary tree search algorithm

Searching algorithms

- Searching for a particular item in a list or a database is a very common operation in computing
- The cards list the top ten most popular girls' names in England in 2015, and the number of babies given each name

Amelia 5,327	Olivia 4,724	Isla 4,012	Emily 3,991	Poppy 3,273
Ava 3,171	Isabella 3,022	Jessica 2,995	Lily 2,965	Sophie 2,905

Searching algorithms

- Using the cards, you can try out two searching algorithms
- Start by putting the cards face down in a line in random order
    - How many babies were named Lily in 2015?

Linear search

- The only systematic way of finding out is to look at each card, starting with the first card, until you
  find Lily
- How many cards did you have to turn up?
- If there are n names in a list, what is the average number of names that will have to be examined?
- What is the “worst case scenario”?

Algorithm for linear search

- Complete this algorithm:

```
function linearSearch(namelist,nameSought)
   index = -1
   i = 0
   found = False
   while i < length(namelist) AND NOT found
      if namelist[i] == nameSought then
         ??????
         ??????
      endif
      ??????
   endwhile
   return index
endfunction
```

Analysing the algorithm

- How many steps are there in the algorithm? What is its time complexity?

```
function linearSearch(namelist,nameSought)
   index = -1
   i = 0
   found = False
   while i < length(namelist) AND NOT found
      if namelist[i] == nameSought then
        index = i
        found = True
      endif
     i = i + 1
   endwhile
   return index
endfunction
```

Big-O for linear search

- There are 2 statements in the loop (an IF statement and an assignment statement) and 3 at the start

```
function linearSearch(namelist,nameSought)
    index = -1
    i = 0
    found = False
    while i < length(namelist) AND NOT found
        if namelist[i] == nameSought then
            index = i
            found = True
        endif
         i = i + 1
    endwhile
    return index
endfunction
```

- Total number of steps = 2n + 3 in worst case
- Time complexity = O(n)

Binary search

- The binary search is a very efficient way of searching a sorted list
    - Examine the middle item in the list
    - If this is the one you are searching for, return the index
    - Eliminate half the list, depending on whether the item being sought is greater than or less than the middle item
    - Repeat until the item is found or is proved to be not in the list

A binary search

- Here is a list of names:

Ali	Ben	Carl	Joe	Ken	Lara	Mo	Oli	Pam	Tara	Stan

The quickest way to find if a particular name is in the list is to do a binary search

- Suppose we are searching for the name Mo
- The list has 11 items
- Examine the middle one first

A binary search

- The middle item in the list is Lara

Ali	Ben	Carl	Joe	Ken	Lara	Mo	Oli	Pam	Tara	Stan

- Lara comes before Mo alphabetically so we can discard all the names from Ali to Lara
- Now we only have 5 names to search

A binary search

- Here is a list of names:

Ali	Ben	Carl	Joe	Ken	Lara	Mo	Oli	Pam	Tara	Stan

- Examine the middle name of the remaining list
- The middle name is Pam
- Mo comes before Pam so we can discard all the names from Pam to Stan

A binary search

- Here is a list of names:

Ali	Ben	Carl	Joe	Ken	Lara	Mo	Oli	Pam	Tara	Stan

- Now we only have two names
- The "middle" name is taken to be the first one
- (e.g. In a list of 6 names, the third name is the middle one)
- Examine the middle name, Mo
  - Bingo! How many names did you look at?

"Divide and conquer"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- In a binary search, the size of the list is approximately halved each time an item is examined
- How many items, at most, would have to be examined in a list of 16 items to find the one you are looking for?
- Try looking for the number 23 in this hidden list of numbers
  - Which box will you look at first?

"Divide and conquer"

							42								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- You're looking for the number 23
- You've found the number 42
    - Which box will you look at next?

“Divide and conquer”

			35												
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- You're looking for the number 25
- You’ve found the number 35
    - Which box will you look at next?

“Divide and conquer”

	27														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- You’ve found the number 27
  - Which box will you look at next?

"Divide and conquer"

23															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- You've found the number 23!
  - How many numbers did you look at?

“Divide and conquer”

23	27	32	35	37	38	41	42	45	50	52	53	54	58	61	67
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

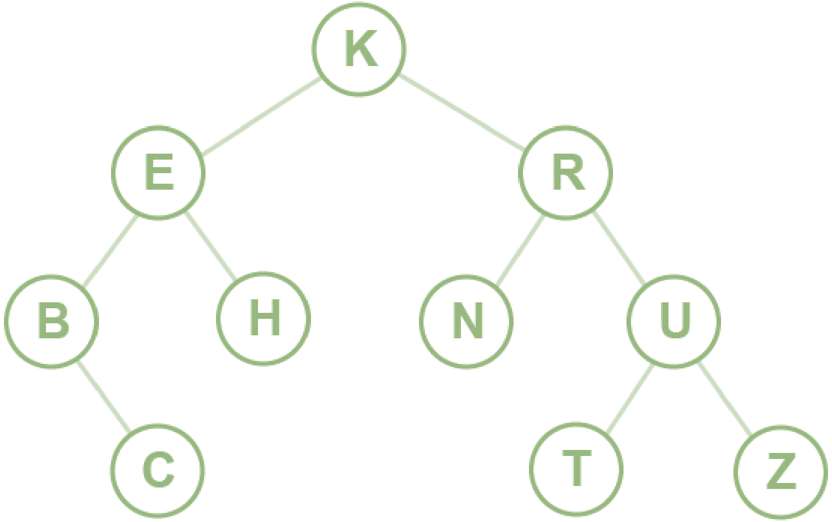
- You looked in boxes 8, 4, 2 and 1
- In a list of 2^n items, the maximum number of items you will need to look at will be *n + 1*
- How many items would be examined if you were searching for 67 instead of 23?
- Try searching for 61 in a list of 15 numbers (delete 67 from the list)
  - How many items need to be examined?

Worksheet 1

- Do the questions in **Task 1**

Binary search trees

- A binary search tree holds items in such a way that the tree can be searched quickly and easily for a particular item
  - Which traversal is used to visit each node in alphabetic sequence?



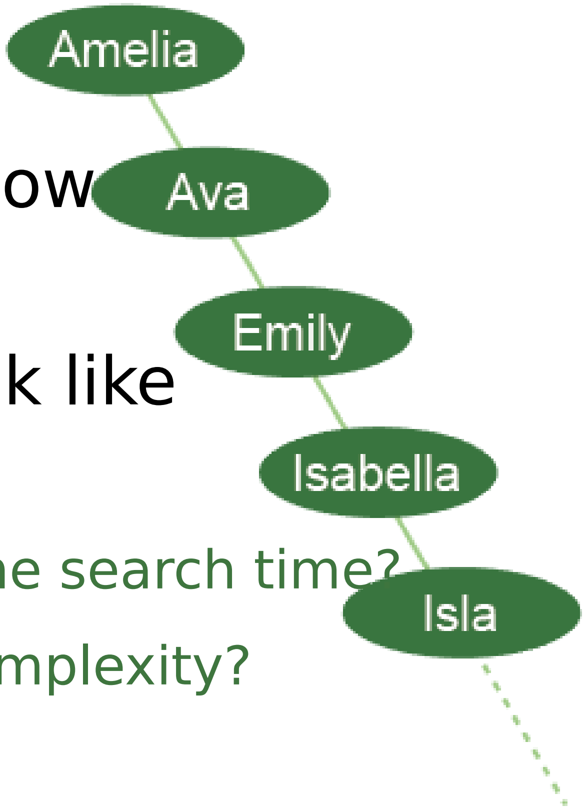
Algorithms for tree traversal

- The algorithm below performs an inorder traversal of the tree

```
procedure traverse(p)
   if tree[p].left != -1 then
      traverse(tree[p].left)
   endif
   print (tree[p].data)
   if tree[p].right != -1 then
      traverse(tree[p].right)
   endif
endprocedure
```

An unbalanced binary tree

- Note that the tree on the previous slide is balanced, as each side has three levels below the root
- An unbalanced tree would look like the one on the right
  - What effect would this have on the search time?
  - What would be the Big-O time complexity?



Worksheet 2

- Do **Task 2** on the worksheet

Plenary

- Three methods of searching which you should be able to explain are:
    - Linear search, binary search, binary tree search
- The time complexity of a linear search is O(n)
- The time complexity of a binary search and binary tree search is O(log n)

PG ONLINE